AFRL-OSR-VA-TR-2015-0022

STRUCTURE AND FUNCTION OF TASK-ORIENTED SOCIAL NETWORKS

Vladimir Filkov
UNIVERSITY OF CALIFORNIA DAVIS

01/05/2015
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTC
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
12-31-2014	Final Report	8/15/11-8/14/14

4. TITLE AND SUBTITLE

STRUCTURE AND FUNCTION OF TASK-ORIENTED SOCIAL NETWORKS

5a. CONTRACT NUMBER
FA9550-11-1-0246

5b. GRANT NUMBER
FA9550-11-1-0246

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Vladimir Filkov

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of California, Davis
One Shields Ave, Davis CA 95616

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This is a final report on our AFOSR grant titled STRUCTURE AND FUNCTION OF TASK-ORIENTED SOCIAL NETWORKS. The goal of the project supported by this grant was to integrate social networks with other empirical data in task oriented projects, in particular Open Source Software projects. Our accomplishments during the three years of the grant are as follows.
- We developed a data framework to gather, store and organize large amounts of publicly available data from Open Source Software projects repositories. All together we have a server with 4TB of structured data from thousands of OSS projects.
- We published 14 publications in top venues in software engineering and computer supported cooperative work. 3 more publications are under review.
- Our publications have won 2 best paper/distinguished paper awards and garnered 3 nominations for best papers.
- We trained 2 postdoctoral scholars, and 1 PhD student.

15. SUBJECT TERMS

Task-Oriented Social Networks, Open Source Software, Task Networks, Software Engineering

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Vladimir Filkov
					19b. TELEPHONE NUMBER *(include area code)* 916-281-6104

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Final Report

for AFOSR Grant STRUCTURE AND FUNCTION
OF TASK-ORIENTED SOCIAL NETWORKS

by Vladimir Filkov

Summary

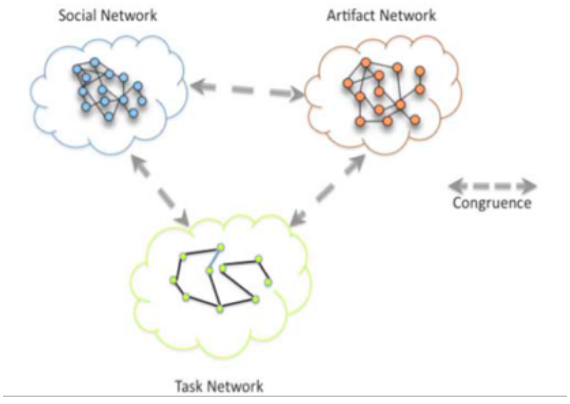
This is the final report on our AFOSR grant titled STRUCTURE AND FUNCTION OF TASK-ORIENTED SOCIAL NETWORKS. The goal of the project supported by this grant was to integrate social networks with other empirical data in task oriented projects, in particular Open Source Software projects. During the three years of the grant (1) we developed a data framework to gather, store and organize large amounts of publicly available data from Open Source Software (OSS) projects repositories; we now have a server with 4TB of structured data from thousands of OSS projects; (2) we developed novel approaches along four main research thrusts: a) metrics for OSS data, b) the social nature of OSS development, c) interaction and dynamics in social task networks, and d) learning in socio-technical environments; (3) we published 14 peer-reviewed publications in top venues in software engineering and computer supported cooperative work. Our publications have been impactful and have attracted attention, including getting 80 citations, winning 2 best paper/distinguished paper awards and garnering 3 nominations for best papers. (4) We also trained 2 postdoctoral scholars, and 2 PhD students.

1. Introduction

a) What We Had Proposed to Do (Summary)

The open, public enactment of open-source software (OSS) development yields valuable data about the performance of self-organizing, distributed, decentralized teams. In our AFOSR grant proposal titled STRUCTURE AND FUNCTION OF TASK-ORIENTED SOCIAL NETWORKS we sought to study the productivity and effectiveness of OSS projects, as exemplar of general task-oriented social networks.

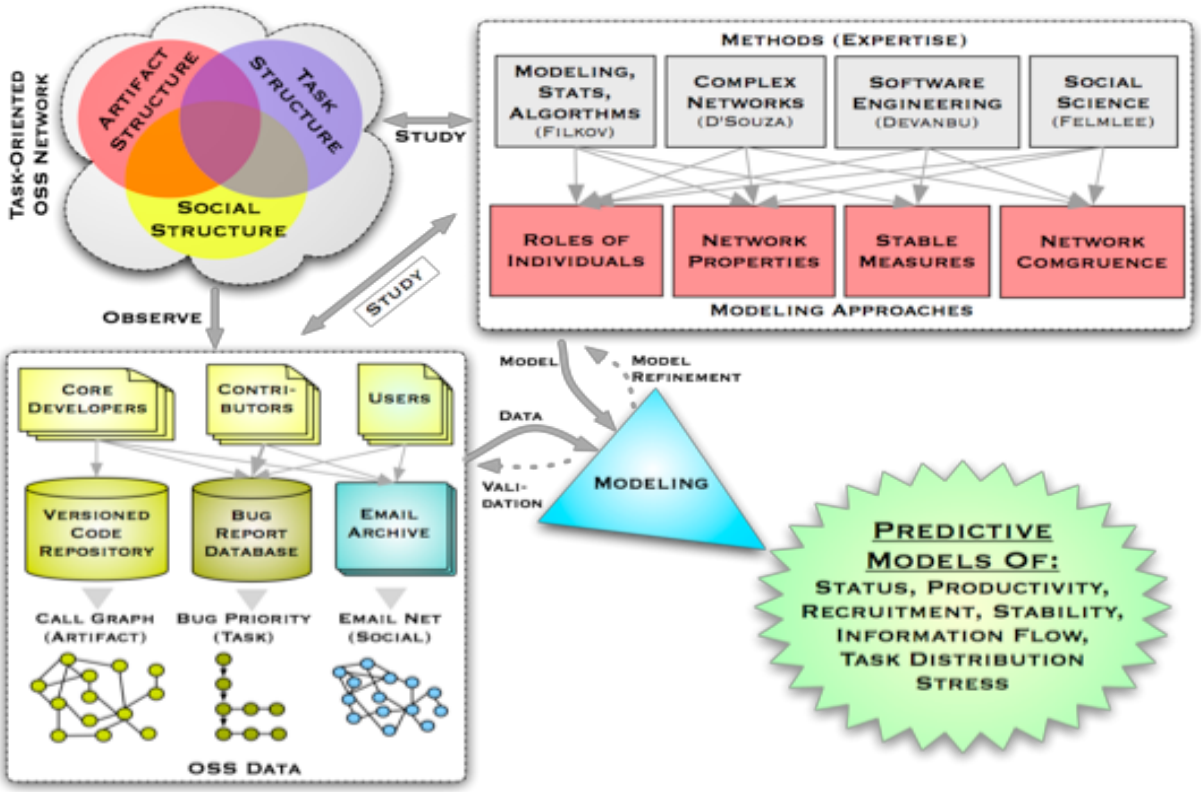
We proposed to develop methods and models that would enable modeling the networks with respect to software engineering outcomes, i.e. predict the effects to the overall efficiency and productivity of developers and code as affected by structural modifications in the networks, social interaction among individuals of different status, interactions between the task structure and the



social network, and the evolution of communities and subcommunities in them and their reaction to stresses. Specifically, our goals were to:

- Devise network analysis measures robust to noise or inadequate observation.
- Mine publicly available open source software project data repositories, including source code repositories, bug reports, and developers email groups to build artifact, task, and social networks.
- Model the structure of a variety of open source software project networks to discover their relationships to status, productivity, recruitment, migration, stress, and others, at the level of individuals, at the level of artifact, task, and social interactions, and finally at the level of overlapping networks.

Our approach had three main strengths: we were to make use of integrated time-series data about open-source projects, which combines individual behavior, emergent network behavior, and task effectiveness. Second, the interdisciplinarity of our team was to allow us to model these complex phenomena using a variety of different models, and validate these models using a variety of techniques. Finally, our strong computational and engineering background was to enable us to use these models to build tools to allow team managers and policy makers to enhance or degrade the performance of task-oriented networks. I summarize our proposed approach in the following figure (copied from the original proposal).

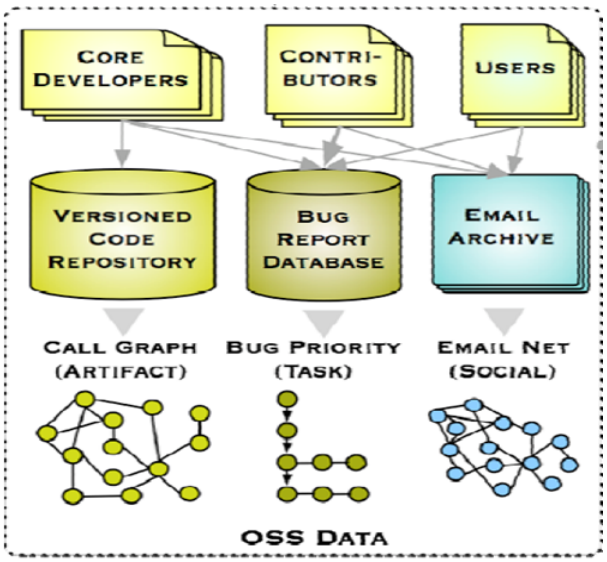


b) What Is in the Rest of This Report

The complete details of our research efforts on this grant are given in the 14 technical papers which have been published over the past 3 years in top venues in computer science, after each underwent extensive peer-review. In the next two sections I provide organized summaries of the content in those papers. The last section of this report contains references to those papers, with URL pointers to their full text.

2. Data Collection and Infrastructure

The Open Source Projects we studied use the versioning repository framework *git* and have trace data publicly available. Their repositories typically have the code, the associated bug lists, and the communication / email archives, from which the structured data we used can be extracted, as shown on the right.



We mined [1,5,6] the Email list archives to extract the senders of messages, the respondents, the time of the message, the content, and threads of discussion. Source code version control repositories were mined [9,11] to provide identity of code changes and content of the change, for every change. From the source code, we also extracted the dependency structure of the system, and track its evolution. We collected bug reports [9] from bug report archives which provide the entire history of each bug report, including priority, severity, and (sometimes) how/where the bug was fixed.

There exist many techniques (some of which have been invented by us) for extracting open-source project data from code repositories. Model version control repositories such as Git not only track the author, details, time, and location of each change; they can also actually very accurately trace the precise provenance (time, and authorship) of each line of code in the repository. This information is readily extracted by processing reports generated by version control systems. Likewise, bug database reports have a regular structure and are easily processed; it also possible, via string matching to identify bug repairs in version control repositories. By tracing the responses to emails, we can identify precise social connections, since the responder has a clearly expressed interest in an email message. Emails do have some irregular structure, but the variations are fairly well understood, and amenable to automatic parsing. A more serious problem is the use of multiple email aliases; we have

developed a fairly scalable, semi-automated approach to detecting aliases [3]. In adversarial settings, relational approach to alias resolution have yielded encouraging results.

We have already a considerable infrastructure of data gathering tools, and we have built up a well-indexed time-series repository of email, bug-report, and version control data from a large number of open-source projects in a variety of domains, including: Apache, PostgresSQL, MySQL, Python, Eclipse, Gimp, Nautilus, Netbeans, and Openoffice. These projects range in size from a few hundred thousand lines of code, to several million lines of code, and involve anywhere from a dozen to several hundred developers. The total data collected is close to 4 Terabytes.

We mined two different collections of OSS projects: the Apache Software Foundation, ASF (http://www.apache.org) from which we collected 136 projects total having all three parts of the data. The second was GitHub (http://www.github.com), whence we extracted a larger number of projects (thousands) but with sparser data for each. From Stack Exchange (http://www.stackexchange.com) and Stack Overflow (http://www.stackoverflow.com) we mined questions that programmers ask about our collected projects, as well as identifying information about the askers and answerers which enabled us to link the GitHub data and Stack Overflow data [3]. The precise details are study specific and are given in the references below.

3. Our Research Contributions

Our research contributions can roughly be categorized in 4 research thrusts: metrics for OSS data, the social nature of OSS development, interaction and dynamics in social task networks, and learning in socio-technical environments. Next we summarize the studies in each.

a. Metrics for OSS Data

The blessing of having collected large amounts of data can quickly turn into a curse if the heterogeneous variance and spurious relationships are not well understood. We spent some time early in the project to investigate good statistical ways of doing initial data exploration.

Better Measures of Correlation (Ref # 15) Empirical software engineering researchers are concerned with understanding the relationships between outcomes of interest, e.g. defects, and process and product measures. The use of correlations to uncover strong relationships is a natural precursor to multivariate modeling. Unfortunately, correlation coefficients can be difficult and/or misleading to interpret. For example, a strong correlation occurs between variables that stand in a polynomial relationship; this may lead one mistakenly, and eventually misleadingly, to model a polynomially related variable in a linear regression. Likewise, a non-monotonic functional, or even non-functional relationship might be entirely missed by a correlation coefficient. Outliers can influence standard correlation measures, tied values can unduly influence even robust non-parametric rank correlation,measures, and smaller sample

sizes can cause instability in correlation measures. A recently developed bivariate measure of association, Maximal Information Coefficient (MIC), promises to simultaneously discover if two variables have: a) any association, b) a functional relationship, and c) a non-linear relationship. The MIC is a very useful complement to standard and rank correlation measures. It separately characterizes the existence of a relationship and its precise nature; thus, it enables more informed choices in modeling non-functional and nonlinear relationships, and a more nuanced indicator of potential problems with the values reported by standard and rank correlation measures. In our first study [15], we illustrated the use of MIC using a variety of software engineering metrics. We studied and explained the distributional properties of MIC and related measures in software engineering data, and illustrated the value of these measures for the empirical software engineering researcher.

Developer focus (Ref # 14) Work practices vary among software developers. Some are highly focused on a few artifacts; others make wide-ranging contributions. Similarly, some artifacts are mostly authored, or "owned", by one or few developers; others have very wide ownership. Focus and ownership are related but different phenomena, both with strong effect on software quality. Prior studies have mostly targeted ownership; the measures of ownership used have generally been based on either simple counts, information-theoretic views of ownership, or social-network views of contribution patterns. We argued [14] for a more general conceptual view that unifies developer focus and artifact ownership. We analogized the developer-artifact contribution network to a predator-prey food web, and drew upon ideas from ecology to produce a novel, and conceptually unified view of measuring focus and ownership. These measures relate to both cross-entropy and Kullback-Liebler divergence, and simultaneously provide two normalized measures of focus from both the developer and artifact perspectives. We argued [14] that these measures are theoretically well-founded, and yield novel predictive, conceptual, and actionable value in software projects. As illustration of the measures' utility, we found that more focused developers introduce fewer defects than defocused developers. In contrast, files that receive narrowly focused activity are more likely to contain defects than other files.

b. Social Nature of OSS Development

(Refs #5, #10) Maintaining a productive and collaborative team of developers is essential to Open Source Software (OSS) success, and hinges upon the trust inherent among the team. Whether a project participant is initiated as a committer is a function of both his technical contributions and also his social interactions with other project participants. One's online social footprint is arguably easier to ascertain and gather than one's technical contributions e.g., gathering patch submission information requires mining multiple sources with different formats, and then merging the aliases from these sources. In this thrust, through two studies [5,10], we analyzed the extent to which one's social activities are predictors for their advancement and status in the OSS communities. In contrast to prior work, where software patch submission was found to be an essential ingredient to achieving committer status, here we investigated the extent to which the likelihood of achieving that status can be modeled

solely as a social network phenomenon. For 6 different Apache Software Foundation OSS projects we compiled and integrated a set of social measures of the communications network among OSS project participants and a set of technical measures, i.e., OSS developers' patch submission activities. We used these sets to predict whether a project participant will become a committer, and to characterize their socialization patterns around the time of becoming committer. We found that the social network metrics, in particular the amount of two-way communication a person participates in, are more significant predictors of one's likelihood to becoming a committer. Further, we found that this is true to the extent that other predictors, e.g., patch submission info, need not be included in the models. In addition, we showed that future committers are easy to identify with great fidelity when using the first three months of data of their social activities. Moreover, only the first month of their social links are a very useful predictor, coming within 10% of the three month data's predictions. Interestingly, we found that on average, for each project, one's level of socialization ramps up before the time of becoming a committer. After obtaining committer status, their social behavior is more individualized, falling into few distinct modes of behavior. In a significant number of projects, immediately after the initiation there is a notable social cooling-off period. Finally, we found that it is easier to become a developer earlier in the projects life cycle than it is later as the project matures. These results should provide insight on the social nature of gaining trust and advancing in status in distributed projects.

c. Interactions and Dynamics in Task Nets

Much of what we do is accomplished by working collaboratively with others, and a large portion of our lives are spent working and talking; the patterns embodied in the alternation of working and talking can provide much useful insight into task-oriented social behaviors. The available electronic traces of the different kinds of human activities in online communities are an empirical goldmine that can enable the holistic study and understanding of these social systems. Open Source Software projects are prototypical examples of collaborative, task-oriented communities, depending on volunteers for high-quality work. In this research thrust, we used longitudinal trace data from hundreds of Apache Software Foundation OSS projects to understand the dynamics and evolution of task networks in open source projects. We leveraged both the social (email communications) and technical (commits to code) contributions of programmers to arrive at an integrated understanding of the socio-technical systems.

Social and Technical Activity Synchronization (Ref #1) To measure the effects of social communications on individuals' working rhythms we developed [1] methods to analyze the communication and code commit records in tens of Open Source Software (OSS) projects. Our methods are based on complex network and time series analysis. We defined the notion of a working rhythm as the average time spent on a commit task and we studied the correlation between working rhythms and communication frequency. We built communication networks for code developers, and found that the developers with higher social status, represented by the nodes with larger number of outgoing or incoming links, always have

faster working rhythms and thus contribute more per unit time to the projects. We also studied the dependency between work (committing) and talk (communication) activities, in particular the effect of their interleaving. We introduced multi-activity time-series and quantitative measures based on activity latencies to evaluate this dependency. Comparison of simulated time-series with the real ones suggested that when work and talk activities are in proximity they may accelerate each other in OSS systems. These findings suggest that frequent communication before and after committing activities is essential for effective software development in distributed systems, and possibly beyond.

Collaborative Synchronization (Refs #4, #11) Synchronized actions are important for completion of complex, interleaved tasks that require the abilities of multiple people [4]. Synchronous development is manifested when file commits by two developers are close together in time and modify the same files. We proposed [11] quantitative methods for identifying synchronized activities in OSS projects, and used them to relate developer synchronization with effective productivity and communication. In particular, we defined co-commit bursts and communication bursts, as intervals of time rich in co-commit and correspondence activities, respectively, and constructed from them smoothed time series which can be, subsequently, correlated to discover synchrony. We found that synchronized co-commits between developers are associated with their effective productivity and coordination: during co-commit bursts, vs. at other times, the project size grows faster even though the overall coding effort slows down. We also found strong correlation between synchronized co-commits and communication, that is, for pairs of developers, more co-commit bursts are accompanied with more communication bursts, and their relationship follows closely a linear model. In addition, synchronized co-commits and communication activities occur very close together in time, thus, they can also be thought of as synchronizing each other. This study can help with better understanding collaborative mechanisms in OSS and the role communication plays in distributed software engineering, and beyond.

Technical Mobility of Software Developers in OSS (Ref #8) Developers in complex, self-organized open-source projects often work on many different files, and over time switch focus between them. Shifting focus can have impact on the software quality and productivity, and is thus an important topic of investigation. We studied [8] the focus shifting patterns (FSPs) of developers by comparing trace data from a dozen open source software (OSS) projects of their longitudinal commit activities and file dependencies from the projects call graphs. Using information theoretic measures of network structure, we found that fairly complex focus shifting patterns emerge, and FSPs in the same project are more similar to each other. We showed that developers tend to shift focus along with, rather than away from, software dependency links described by the call graphs. This tendency becomes weaker as either the interval between successive commits, or the organizational distance between committed files (i.e. directory distance), gets larger. Interestingly, this tendency appears stronger with more productive developers. We hope our study will initiate interest in further understanding of FSPs, which can ultimately help to (1) improve current recommender

systems to predict the next focus of developers, and (2) provide insight into better call graph design, so as to facilitate developers' work.

Task Networks Communities of Like-Culture (Ref #15) We looked [15] at the emergence of larger, team-level, collaborative structures in task networks. We used sequence analysis methods to identify the work-talk patterns of software developers in these online communities. We found that software developers prefer to persist in same kinds of activities, i.e., a string of work activities followed by a string of talk activities and so forth, rather than switch them frequently; this tendency strengthens with time, suggesting that developers become more efficient, and can work longer with fewer interruptions. This process is accompanied by the formation of community culture: developers' patterns in the same communities get closer with time while different communities get relatively more different. The emergence of community culture is apparently driven by both "talk" and "work". Finally, we also found that workers with good balance between "work" and "talk" tend to produce just as much work as those that focus strongly on "work"; however, the former appear to be more likely to continue to be active contributors in the communities.

d. Learning the programming and social environments

In this research thrust we sought to understand how developers use existing code and how they learn to use existing and new code. Stack Exchange is a very popular Question & Answer internet community. Users can post questions on a wide variety of topics; other users provide answers, usually within minutes. Participants are not compensated for their services and anyone can freely gain value from the efforts of the users; Stack Exchange is therefore a gift economy. Users, however, do gain reputation points when other users "upvote" their questions and/or answers. Stack Exchange thus functions as a learning community with a strong reputation-seeking element that creates a valuable public good, viz., the Q&A archive.

Where Do OSS Coders Learn From? (Ref #3) Stack Overflow, a part of Stack Exchange, is a popular online programming question and answer community providing its participants with rapid access to knowledge and expertise of their peers, serving programmers and coders. Despite the popularity of Stack Overflow, its role in the work cycle of open-source developers is yet to be understood: on the one hand, participation in it has the potential to increase the knowledge of individual developers thus improving and speeding up the development process. On the other hand, participation in Stack Overflow may interrupt the regular working rhythm of the developer, hence also possibly slow down the development process. We investigated [3] the interplay between Stack Overflow activities and the development process, reflected by code changes committed to the largest social coding repository, GitHub. Our study showed that active GitHub committers ask fewer questions and provide more answers than others. Moreover, we observed that active Stack Overflow askers distribute their work in a less uniform way than developers that do not ask questions. Finally, we showed that despite the interruptions incurred, the Stack Overflow activity rate correlates with the code changing activity in GitHub.

What Do OSS Coders Learn About? (Ref #7) Programming is knowledge intensive. While it is well understood that programmers spend lots of time looking for information, with few exceptions, there is a significant lack of data on what information they seek, and why. Modern platforms, like Android, comprise complex APIs that often perplex programmers. We asked [7]: which elements are confusing, and why? Increasingly, when programmers need answers, they turn to Stack Overflow. This provides a novel opportunity. There are a vast number of applications for Android devices, which can be readily analyzed, and many traces of interactions on Stack Overflow. These provide a complementary perspective on using and asking, and allow the two phenomena to be studied together. How does the market demand for the USE of an API drive the market for knowledge about it? We analyzed [7] data from Android applications and Stack Overflow together, to find out what it is that programmers want to know and why.

Who Are the Teachers? (Ref #2) The incentive structure of the Stack Overflow community suggests that over time, the quality of the product (viz.., delivered answers) steadily improves, and furthermore, that any individual who durably participates in this community for an extended period also would enjoy an increase in the quality of their output (viz., the answers they provide). We investigated [2] the validity of these widely held beliefs in greater detail, using data downloaded from Stack Exchange. Our analysis indicates that these intuitions are actually not supported by the data; indeed the data suggests that overall answer scores decrease, and that people's tenure with the community is unrelated to the quality of their answers. Most interestingly, we show that answering skill, i.e. getting high average answer scores, which is different than reputation, is evident from the start and persists during one's tenure with the community. Conversely, people providing low rated answers are likely to have done so from the start.

Learning in Socio-Technical Environments (Ref #12) Historically, mailing lists have been the preferred means for coordinating development and user support activities. With the emergence and popularity growth of social Q&A sites such as the Stack Exchange network (e.g., Stack Overflow), this is beginning to change. Such sites offer different socio-technical incentives to their participants than mailing lists do, e.g., rich web environments to store and manage content collaboratively, or a place to showcase their knowledge and expertise more visibly to peers or potential recruiters. A key difference between Stack Exchange and mailing lists is gamification, i.e., Stack Exchange participants compete to obtain reputation points and badges. Using a case study of R, a popular data analysis software, we investigated [12] how mailing list participation has evolved since the launch of StackExchange. Our main contribution is assembling a joint data set from the two sources, in which participants in both the r-help mailing list and Stack Exchange are identifiable. This allows for linking their activities across the two resources and also over time. With this data set we found that user support activities are showing a strong shift away from r-help. In particular, mailing list experts are migrating to Stack Exchange, where their behavior is different. First, participants active both on r-help and on Stack Exchange are more active than those who focus exclusively on only one of the two. Second, they provide faster answers on Stack Exchange than on r-help,

suggesting they are motivated by the gamified environment. To our knowledge, our study is the first to directly chart the changes in behaviour of specific contributors as they migrate into gamified environments, and has important implications for knowledge management in software engineering.

4. Our Publications Supported by This Grant and URLs to Their Full Text

[1] Qi Xuan, Mohammad Gharehyazie, Premkumar Devanbu, Vladimir Filkov. Measuring the Effect of Social Communications on Individual Working Rhythms: A Case Study of Open Source Software. ASE/IEEE International Conference on Social Informatics, 2012

http://web.cs.ucdavis.edu/~filkov/papers/rhythm.pdf

[2] D. Posnett, E. Warburg, P. Devanbu, V. Filkov. Mining Stack Exchange: Expertise is Evident From Initial Contributions. ASE/IEEE Interntl. Conference on Social Informatics, 2012

http://web.cs.ucdavis.edu/~filkov/papers/stex.pdf

[3] Bogdan Vasilescu, Vladimir Filkov, Alexander Serebrenik. StackOverflow and GitHub: Associations Between Software Development and Crowdsourced Knowledge. IEEE International Conference on Social Computing (SocialCom 2013)

http://web.cs.ucdavis.edu/~filkov/papers/socialcom.pdf

[4] Qi Xuan, Vladimir Filkov. Synchrony in Social Groups and Its Benefits. Chapter in Handbook of Human Computation, Michelucci P. (ed.), Springer 2013

http://web.cs.ucdavis.edu/~filkov/papers/synchrony.pdf

[5] Mohammad Gharehyazie, D. Posnett, V. Filkov. Social Activities Rival Patch Submission For Prediction of Developer Initiation in OSS. Projects Nominee: ACM Distinguished Paper Award. 29th IEEE International Conference on Software Maintenance, ICSM 2013

http://web.cs.ucdavis.edu/~filkov/papers/socialvspatch.pdf

[6] Daryl Posnett, Raissa D'Souza, Prem Devanbu, Vladimir Filkov. Dual Ecological Measures of Focus for Software Development. Winner: ACM Distinguished Paper Award. ACM/IEEE 35th International Conference of Software Engineering, ICSE 2013

http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6606591&queryText%3Dfilkov

[7] David Kavaler, D. Posnett, Clint Gibler, Hao Chen, Prem Devanbu, Vladimir Filkov. Using and Asking: APIs Used in the Android Market and Asked About in StackOverflow. Best Paper Runner-Up (top 5%) The 5th International Conference on Social Informatics (SocInfo2013)

http://web.cs.ucdavis.edu/~filkov/papers/usingandasking.pdf

[8] Qi Xuan, Aaron Okano, Prem Devanbu, Vladimir Filkov. Focus-Shifting Patterns of OSS Developers and Their Congruence with Call Graphs. ACM SIGSOFT FSE 2014

http://web.cs.ucdavis.edu/~filkov/papers/focusshift.pdf

[9] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, Prem Devanbu. A Large Scale Study of Programming Languages and Code Quality in Github. ACM SIGSOFT FSE 2014

http://web.cs.ucdavis.edu/~filkov/papers/lang_github.pdf

[10] Mohammad Gharehyazie, Daryl Posnett, Bogdan Vasilescu, Vladimir Developer initiation and social interactions in OSS: A case study of the Apache Software Foundation. Empirical Software Engineering 2014, 1-36

http://web.cs.ucdavis.edu/~filkov/papers/developer_trust.pdf

[11] Qi Xuan, Vladimir Filkov. Building It Together: Synchronous Development in OSS. ACM/IEEE 36th International Conference on Software Engineering, ICSE 2014

http://web.cs.ucdavis.edu/~filkov/papers/together.pdf

[12] Bogdan Vasilescu, A. Serebrenik, Prem Devanbu, Vladimir Filkov. How Social Q&A Sites are Changing Knowledge Sharing in Open Source Software Communities. 17th ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW 2014)

http://web.cs.ucdavis.edu/~filkov/papers/r_so.pdf

[13] D. Posnett, V. Filkov, P. Devanbu. Ecological inference in empirical software engineering. Winner: Best Paper Award and ACM Distinguished Paper Award. Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2011

http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6100074

[14] D. Posnett, P. Devanbu, V. Filkov. MIC Check: A Correlation Tactic for ESE Data. Proc. of the 9th IEEE/ACM Working Conference on Mining Software Repositories (MSR), 2012
http://web.cs.ucdavis.edu/~filkov/papers/mic-check.pdf

[15] Qi Xuan, Prem Devanbu, Vladimir Filkov. Converging Work-Talk Patterns in Online Task-Oriented Communities. CoRR abs/1404.5708 (2014) (under review)

http://arxiv.org/abs/1404.5708